FINAL

First Code Improvement Completed

Numerical Simulations For Active Tectonic Processes: Increasing Interoperability And Performance

JPL Task Order: 10650

Milestone F – Code Improvement due date: 6/30/2003

First code improvement (functional enhancement and speedup). Documented source code made publicly available via the Web.

- PARK on 256 CPU machine with 150,000 elements, 5,000 time steps in the same time as the baseline case
- GeoFEST - links to PYRAMID and runs on a parallel machine - Produce a plot of scaled speedup that will show that we are maintaining efficiency as the number of processors and problem size increase. Assuming availability of a 64 CPU Beowulf, 1,250,000 elements, 1000 timesteps, in the same time as the baseline case.

Team

Andrea Donnellan:
Principal Investigator

Jet Propulsion Laboratory
Mail Stop 183-335
4800 Oak Grove Drive
Pasadena, CA 91109-8099
donnellan@jpl.nasa.gov
818-354-4737

Terry Tullis:
Fast Multipole Methods

Brown University
Box 1846, Brown University
Providence, RI 02912-1846
Terry_Tullis@Brown.edu
401-863-3829

Jay Parker:
Overall Software Engineer

Jet Propulsion Laboratory
Mail Stop 238-600
4800 Oak Grove Drive
Pasadena, CA 91109-8099
Jay.W.Parker@jpl.nasa.gov
818-354-6790

Geoffrey Fox:
Information Architect

Community Grid Computing Laboratory
Indiana University
501 N. Morton, Suite 224
Bloomington, IN 47404-3730
gcf@indiana.edu
812-856-7977

Dennis McLeod:
Database Interoperability

Professor
Computer Science Department
University of Southern California
Los Angeles, CA 90089-0781
mcleod@usc.edu
213-740-4504

John Rundle:
Pattern Recognizers

Center for Computational Science and Engineering
U. C. Davis
Davis, CA 95616
rundle@geology.ucdavis.edu
530-752-6416

Greg Lyzenga:
Finite Element Models

Jet Propulsion Laboratory
Mail Stop 126-347
4800 Oak Grove Drive
Pasadena, CA 91109-8099
greg.lyzenga@jpl.nasa.gov
818-354-6920

Michele Judd:
Technical Task Manager

Jet Propulsion Laboratory
Mail Stop 183-335
4800 Oak Grove Drive
Pasadena, CA 91109-8099
michele.judd@jpl.nasa.gov
818-354-4994

Marlon Pierce:
Code Interoperability Software Engineer

Community Grid Computing Lab
Indiana University
501 N. Morton, Suite 224
Bloomington, IN 47404-3730
marpierc@indiana.edu
812-856-1212

Lisa Grant:
Fault Database Architect

University of California, Irvine
Environmental Analysis and Design
Irvine, CA 92697-7070
lgrant@uci.edu
949-824-5491

Robert Granat:
Pattern Recognizers

Jet Propulsion Laboratory
Mail Stop 126-347
4800 Oak Grove Drive
Pasadena, CA 91109-8099
robert.granat@jpl.nasa.gov
818-393-5353

Teresa Baker:
GeoFEST Code Verification

Jet Propulsion Laboratory
Mail Stop 300-233
4800 Oak Grove Drive
Pasadena, CA 91109-8099
teresa.baker@jpl.nasa.gov
818-354-4350

Overview

This milestone report documents Milestone F of the QuakeSim project (Numerical Simulations For Active Tectonic Processes: Increasing Interoperability And Performance) for NASA's Earth Science Technology Office, Computational Technology Program. The text of the milestone appears on Page 1, and requires demonstration of substantially larger problems than Milestone E (7/30/02, Table 1) for the now-parallel earthquake codes PARK and GeoFEST. Milestone F also requires demonstration of parallel scaling.

In the next section we describe the large problems that demonstrate code improvement for the PARK and GeoFEST code. Then we give details for the PARK code (code description, algorithm, numerical method, documentation, scaling analysis, and scientific and computational significance), and for the GeoFEST code (including the same topics).

Code	Machine Wallclock Time	Processors	Date	Elements	Time Steps
PARK Milestone E (7/30/02)	*Chapman (AMES)* 7.888 Hours	1	September 18, 2002	15,000	500
PARK Milestone F	*Chapman* (AMES) 7.879 Hours	256	August 15, 2003	150,000	5,000
GeoFEST Milestone E (7/30/02)	Solaris workstation (JPL) 13.7 Hours	1	July 30, 2002	55,369	1,000
GeoFEST Milestone F	*Thunderhead* (GSFC) 2.8 Hours	64	September 1, 2003	1,400,198	1,000

Table 1: Computer runs demonstrating baseline and Milestone F performance enhancements for PARK and GeoFEST applications.

Problems Used to Demonstrate Code Improvement

Table 1 summarizes the results for the improved code demonstration runs. Additional runs demonstrating parallel scaling are detailed in Section 4.

The nature of the code improvements we describe is that problems of much larger size than the baseline case are solved in the same time, by use of advanced computing technology (Multipole methods for PARK, domain partitioning for GeoFEST, and efficient MPI parallel coding for both PARK and GeoFEST).

For the PARK code the problem demonstrating the improvement over baseline has geometry shown in Figure 1. The boundary conditions (appropriate for the geographic setting at Parkfield, CA) are the same as for the baseline case, but the mesh density is increased from 15,000 to 150,000 rectangular elements, and the problem duration is extended from 500 to 5000 time steps. Traditional methods lead to work scaling with the square of the number of elements, and linear with time steps. Our improvements are due to efficient parallel implementation and use of a Multipole solution technique that scales much better than a law quadratic in element count.

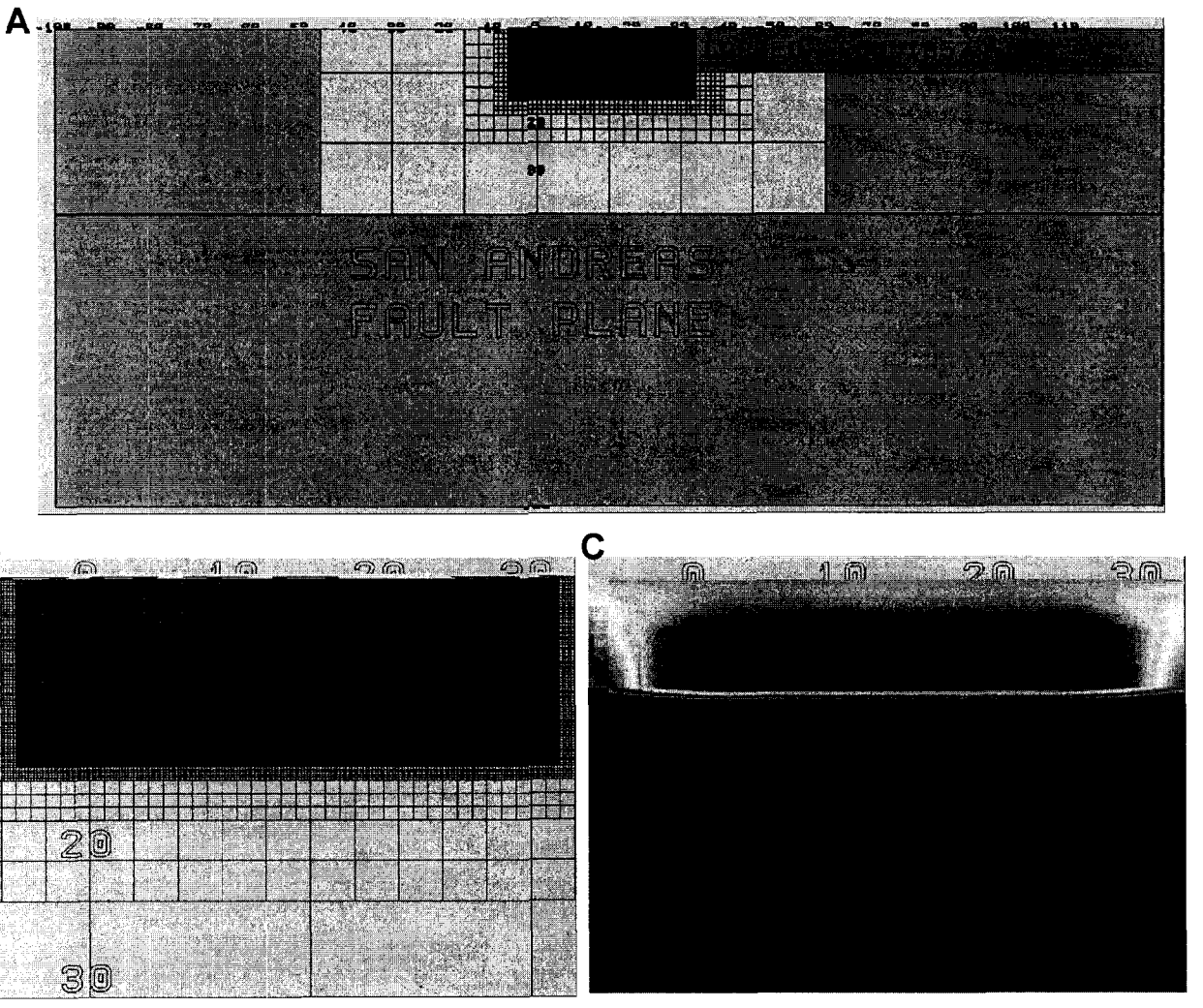

Figure 1. Views of grid geometry and distribution of constitutive properties for PARK at three different scales.

A. Shows the overall geometry. The numbers are distances in km, both vertically and horizontally from the origin, which is at the earth's surface at Middle Mountain, directly above the hypocenter of actual Parkfield earthquakes. Boundary conditions of 35 mm/yr slip rate are applied on the light brown areas, while the blue area is locked with zero slip rate.

B. Closer view of the grid geometry. The large nearly solid red area is a region with square elements ~65 m on a side. Within it is an area at about 10 km deep and 3 km horizontally that appears solid red at this magnification, with ~22 m squares and ~7.4 m squares.

C. Same area as in (**B**) showing contours of the constitutive parameter *a-b* without the grid. Instability can occur in areas colored brown, the darkness being proportional to the tendency to be unstable.

D. Magnification of the finest area, showing the ~7.4 m elements and part of the area with ~22 m elements. As in (**C**) the color shows the intensity of *a-b*.

For GeoFEST, the problem geometry is shown in Figure 2. Elastic and viscoelastic deformation is simulated for 1.4 million finite elements, for a duration of 1000 time steps. The significant increase is in the number of elements, which was about 55,000 for the baseline case (also running 1000 time steps). Note that the baseline case was modeled on the 1994 Northridge earthquake, using a single fault of 300 square km in a domain of 240 x 240 x 100 km. The improved demonstration case is based on the 1992 Landers event, using three closely arranged faults within an 865 square km area in a domain volume area of 1000 x 1000 x 60 km.

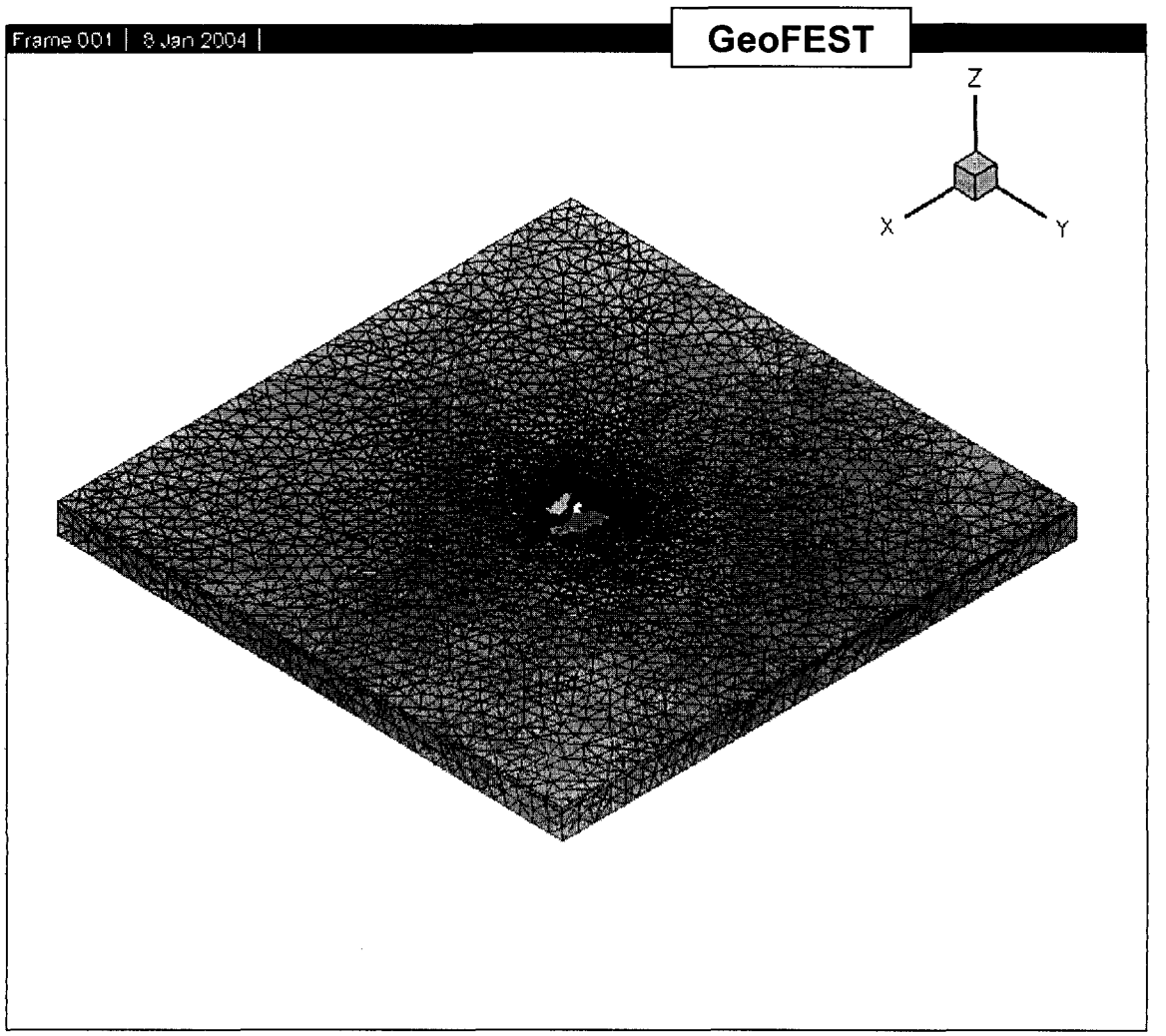

Figure 2a: Finite element mesh LandersGap64 for GeoFEST milestone F improvement code improvement problem. Colors indicate partitioning among processors (limited to 16 processors in this image for clarity, actually 64 processors were used). Partitions cluster near domain center due to the high mesh density that is used near the faults.

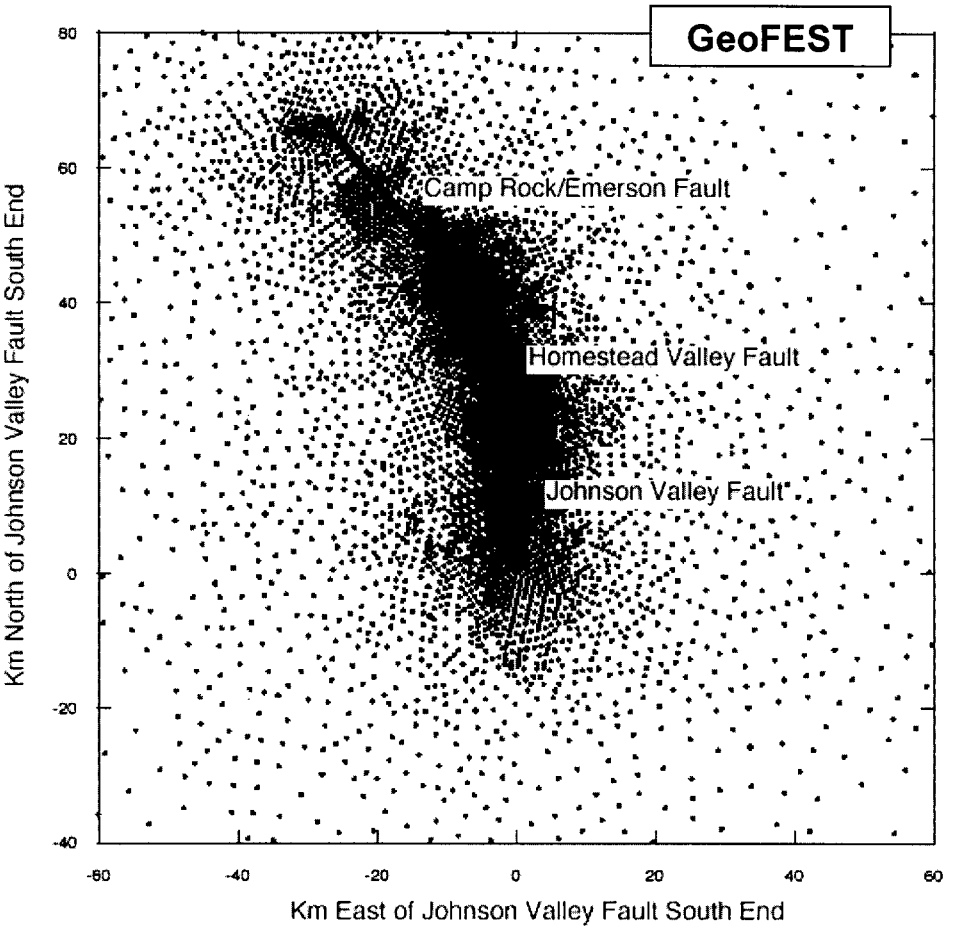

Figure 2b: Fault segments and surface nodes for LandersGap64 mesh, center region.

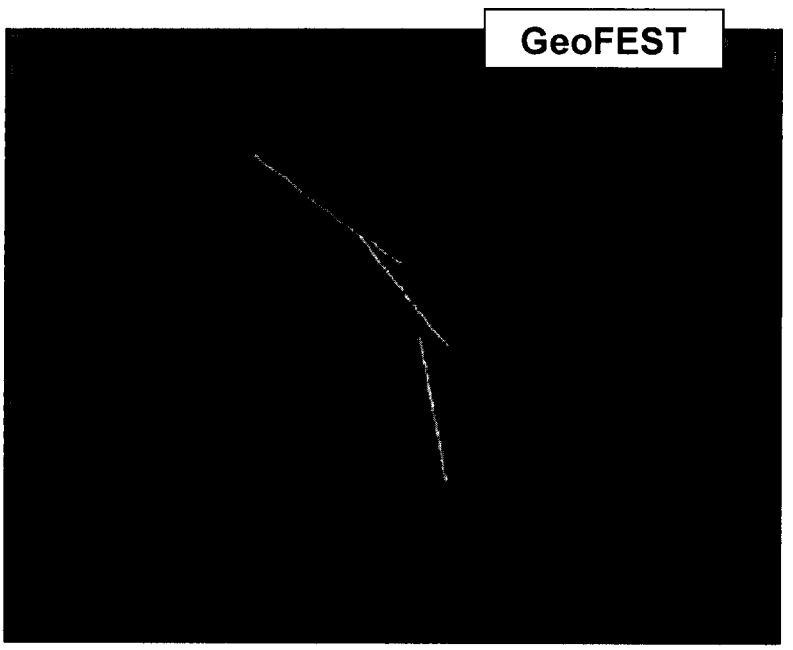

Figure 2c:

GeoFEST simulated surface displacement from coseismic Landers model, displayed as InSAR fringes (5.2 cm vertical displacment is one color cycle).

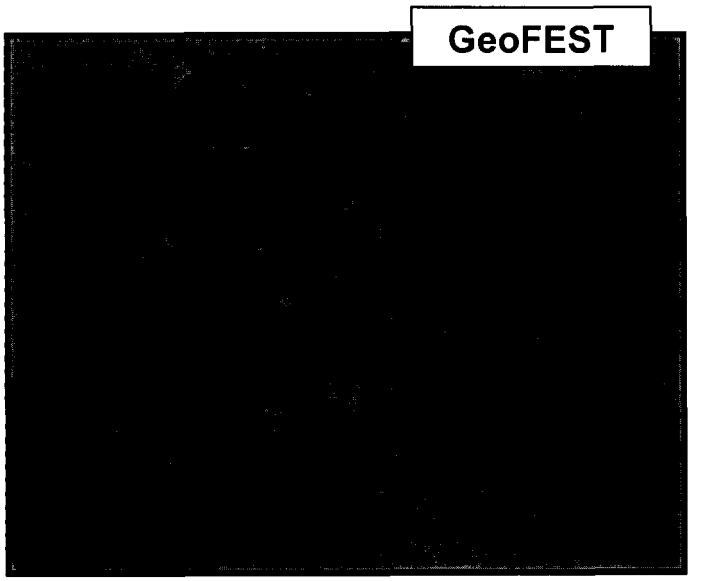

Figure 2d:

GeoFEST simulated postseismic surface displacement from Landers model after 500 years of viscoelastic relaxation (at the end of the GeoFEST Milestone F case of Table 1). Color scale of InSAR fringes is that of Figure 2c.

Supporting Documents

The top-level web site for the QuakeSim task is at http://quakesim.jpl.nasa.gov. Source code for the three codes may be found at http://quakesim.jpl.nasa.gov/download.html. Files required for the baseline cases may be found at http://quakesim.jpl.nasa.gov/milestones.html.

The PARK Code

PARK is a model for unstable slip on a single earthquake fault. Because it aims to capture the instability, it is designed to represent the slip on a fault at many scales, and to capture the developing seismic slip details over an extraordinary range of time scales (sub-seconds to decades). Its simulation of the evolution of fault rupture is the most realistic of the tools in QuakeSim for that scale, demonstrating the multi-scale approach of this project. When transformed into an efficient parallel simulation, it will be a powerful tool for researchers seeking to determine the nature and detectability of earthquake warning signals such as surface strains and patterns of microseismicity.

In a typical application the PARK code will compute the history of slip, slip velocity, and stress on a vertical strike-slip fault that results from using state-of-the-art rate and state frictional constitutive laws on the fault, which is currently that for a specific geographic setting at Parkfield, California. The boundary conditions are those appropriate for Parkfield, and the distribution of constitutive properties on the fault zone are as realistic as our ability to characterize the subsurface properties of the fault there allows. The methods developed in solving this problem can be generalized to other geologic settings in which the fault geometry and the boundary conditions are not so simple and multiple faults are

involved. The fault is represented by a rectangular grid with highly variable mesh density (Figure 1).

Algorithm

The main program is a boundary element program that determines the stress on every element of the fault surface due to slip on every other element, using a Greens function approach. The fault constitutive law is used to determine what the slip velocity will be for that stress. The velocity multiplied by the time step gives the slip for calculating the stress in the next time increment. This involves the forward time integration of coupled ordinary differential equations.

Numerical Methods

Numerically, PARK is a boundary element program that determines the stress on every element of the fault surface due to slip on every other element, using a Greens function approach. This is the first earthquake simulation code to seek and achieve enhanced scalability and speed by employing a Multipole technique (documented below). The Multipole experience gained here will also be transferable to the Virtual California code and other boundary element simulations. The power of massive parallel computing is required for this problem in order to support many small slip patch elements needed to cover the nucleation scale that initiates the instability.

The integration is done with a fifth order Runge-Kutta[1] scheme with adaptive step size control. Because the time-steps range over ten orders of magnitude, the adaptive step-size control is an essential element in the solution. The time steps range depending on whether the fault is slipping very slowly in the interseismic period or very fast during an earthquake.

The main program calls a variety of subroutines and the one of these subroutines that calculates the derivatives used in the forward time integration itself calls a Fast Multipole library that is suitable for such Green's functions problems. The Multipole approach allows a number of computations to scale as N log N rather than N^2 as would otherwise be the case. This Fast Multipole approach allows determination of the degree of grouping of the remote cells based on an analytical approximation to the Green's function. In order to reduce computation time it also renumbers the elements so that those that are near in space are also near in memory.

---

[1] Runge-Kutta is a method for forward integration of differential equations that involves calculating derivatives of the functions at the current time and several fractions of potential time-steps in the future, appropriately weighting these derivatives estimating the best derivative value to use and determining the value of the function at the new time by multiplying that best derivative by the appropriate time-step. The fifth-order Runge Kutta method compares estimates made using two different time-steps and, based on this comparison, determines whether a smaller or larger time-step should be used for the next step. This allows for adaptive time-stepping which is extremely important in problems such as this where the time-steps can vary as much as ten orders of magnitude, depending on whether interseismic or a coseismic behavior is involved.

Parallel Implementation

The flow of the program and the functions of its main routines are found in the file Code_Description.doc posted at http://quakesim.jpl.nasa.gov/milestones and also http://quakesim.jpl.nasa.gov/documentation. The main program and most of its subroutines are written in FORTRAN 90. These programs were converted to use MPI to run in parallel for the First Code Improvement Milestone.

The main program reads inputs and sets up the problem, carries out a time-stepping loop that integrates the slip and stress field forward in time, and writes a summary and closes files at the end. The dominant task is a function "DERIVS," which uses a fault constituitive relationship to determine slip velocity based on local stress, includes effects of shear wave speed and radiation damping, and uses the Multipole library to sum the stress contributions of the slip on all fault elements to determine the local stress. So, within a single time step the objective is to integrate the history using variable-step Runge-Kutta based on the stress-velocity interactions. In each time step "DERIVS" is called a total of six times, once initially by the main program and five more times by the integrator, to result in a fifth-order Runge Kutta integration. "DERIVS" relies on the Multipole code so it calls "sumtree" to gather all the contributions to local stress, and then determines the local stress and slip temporal derivatives from that. Finally it calls "forgettree" to clear the Multipole structure.

In the parallel version, a single processor reads the geometry and partitions it among the remaining processors, sending a portion of the geometry and local properties to each one. Within the loop over time steps "DERIVS" uses the Mulipole functions similar to the sequential code, except that "sumtree" is designed for parallel use and has internal MPI calls.

The function computing the stress interactions between fault elements dominates the work in the problem. This interaction is made fast and efficient by using a proven parallel Fast Multipole library. Parallel decomposition was added to portions of the remaining code in order to make good use of this parallel library.

We link the Fast Multipole library of Salmon and Warren (Salmon and Warren, 1997; Warren and Salmon, 1997), which is written in parallel using MPI.

Documentation

Software documentation is summarized in this report and can be found, in full, in the subdirectories of the 1st_Code_Improv_Milestone directory that is accessible at the public URL http://www.servogrid.org/slide/GEM/PARK.

For the purpose of verifying that the First Code Improvement Milestone run is as described in the "Milestone_Certification_Data" file and repeating the run if desired, the compiled files and documentation are available on turing, an SGI Origin 3000 at NASA Ames. This pre-compiled version, on a NASA Ames machine, uses two copyrighted numerical recipes subroutines, which are

available in the **src-bin** subdirectory of **turing:/u/tullis/1st_Code_Improv_Milestone**. The subroutines are easily and inexpensively available, however they cannot be posted on the public web site.

Source code and files needed to compile PARK, as well as input files for verification of the First Code Improvement run have been placed in unix-compressed tar files available in two locations. The first is a website (http://www.servogrid.org/slide/GEM/PARK/1st_Code_Improv_Milestone/Downloads) and the second is two of the SGI Origin 3000s at NASA Ames (turing or "chapman:/u/tullis/1st_Code_Improv_Milestone/Downloads"). On the Origin 3000s, the directory Downloads contains two files, "PARK_Package_1st_Improv.tar.Z" and "PARK_Package_NR.tar.Z". The files are identical except for the two copyrighted numerical recipes subroutines, which are only included in the "PARK_Package_NR.tar.Z" version on *turing*. Except for the presence or absence of these two subroutines, both of these tar files will create the Multipole library, the source files for the PARK fault application, and the input files for the First Code Improvement Milestone run. On the website, only the "PARK_Package_1st_Improv.tar.Z" file exists, so that the copyrighted subroutines are not made public. See either the "README-src-bin.txt" file in the src-bin directory or the header for the "park.f" file to learn what needs to be done to create these numerical recipes subroutines.

When the tar files are extracted and decompressed, compiling the libraries and the executable will create two files in the **t17-7/Objfiles/IRIX64** directory, "libsw.a" and "mpmy_seq.o". In addition, object files and the executable file ("park") will be created in the **src-bin** directory.

Scaling Analysis

In the directory **scaling**, found in the same **1st_Code_Improv_Milestone** directory in which this file is found, there are a number of files that show how the job scales both with the number of elements and the number of processors. Thirty-six scaling runs were done, involving all the combinations of the number of elements used (712, 5,292, 15,000, and 150,000) and number of processors used (1, 2, 4, 8, 16, 32, 64, 128, and 256) (Table 2, Figure 3). All these scaling runs were run for 100 time steps, whereas the full 150,000 element, 256 processor First Code Improvement Milestone run was done for 5,000 time steps. In the **scaling** directory is a data table giving the walltime for all 36 scaling runs, as well as five plots showing dependence of walltime, speedup, efficiency, and overhead on number of elements and number of processors.

Data for Scaling Runs for PARK						
nprocs	elements	time steps	walltime	idn	iremain	elements/nproc
1	712	100	0:01:13	0	712	712
2	712	100	0:01:01	357	355	356
4	712	100	0:00:48	179	175	178
8	712	100	0:00:41	90	82	89
16	712	100	0:00:41	45	37	45
32	712	100	0:00:48	22	30	22
64	712	100	0:01:11	11	19	11
128	712	100	0:01:39	5	77	6
256	712	100	0:01:45	2	202	3
1	5292	100	0:11:33	0	5,392	5,292
2	5292	100	0:08:58	2,697	2,695	2,646
4	5292	100	0:05:56	1,349	1,345	1,323
8	5292	100	0:03:48	675	667	662
16	5292	100	0:02:07	338	322	331
32	5292	100	0:01:34	169	153	165
64	5292	100	0:01:42	85	37	83
128	5292	100	0:02:28	42	58	41
256	5292	100	0:03:53	21	37	21
1	15000	100	0:32:12	0	15,000	15,000
2	15000	100	0:29:57	7,501	7,499	7,500
4	15000	100	0:15:00	3,751	3,747	3,750
8	15000	100	0:09:51	1,876	1,868	1,875
16	15000	100	0:05:17	938	930	938
32	15000	100	0:02:56	469	461	469
64	15000	100	0:02:18	235	195	234
128	15000	100	0:02:37	118	14	117
256	15000	100	0:04:24	58	210	59
1	150000	100	7:57:47	0	150,000	150,000
2	150000	100	7:41:56	75,001	74,999	75,000
4	150000	100	4:02:22	37,501	37,497	37,500
8	150000	100	2:01:39	18,751	18,743	18,750
16	150000	100	1:17:24	9,376	9,360	9,375
32	150000	100	0:38:39	4,688	4,672	4,688
64	150000	100	0:20:48	2,344	2,328	2,344
128	150000	100	0:12:12	1,172	1,156	1,172
256	150000	100	0:09:27	586	570	586

Table 2: Table 2 shows the problem sizes and run times of PARK on *chapman* using several different numbers of processors. In order to make this scaling study, the milestone problem was reduced to 100 time steps (rather than 5,000) These numbers form the basis for all the plots shown in Figure 3.

Figures 3a, 3b, 3c (below): Several ways of presenting how run time varies with problem size and number of processors for PARK on *chapman* (Table 2).

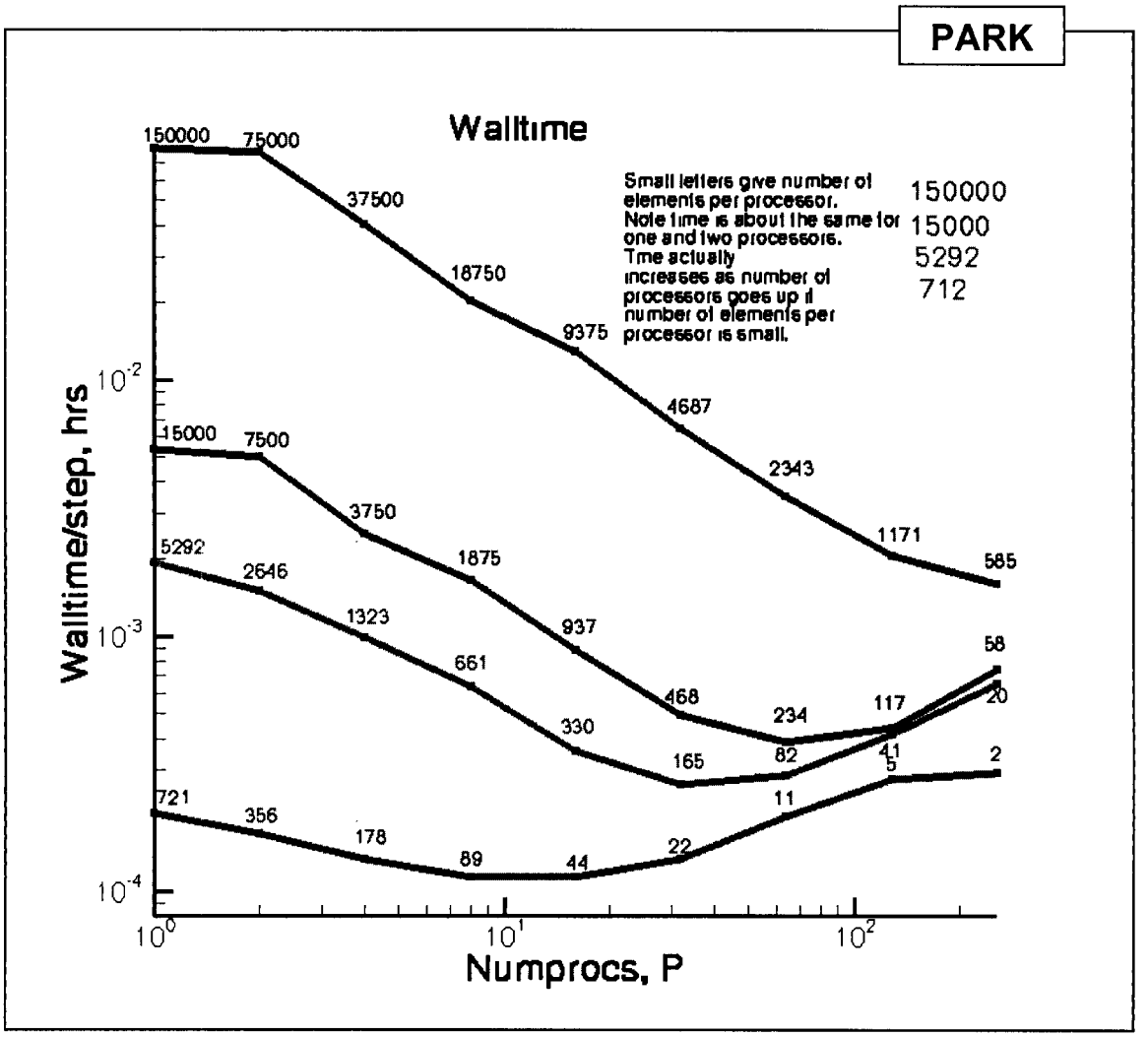

Figure 3a. PARK: Walltime. Generally we find that problems complete faster when more processors are applied. Very small problems take longer to run on many processors, and moderate problems have an optimal machine size that should not be exceeded.

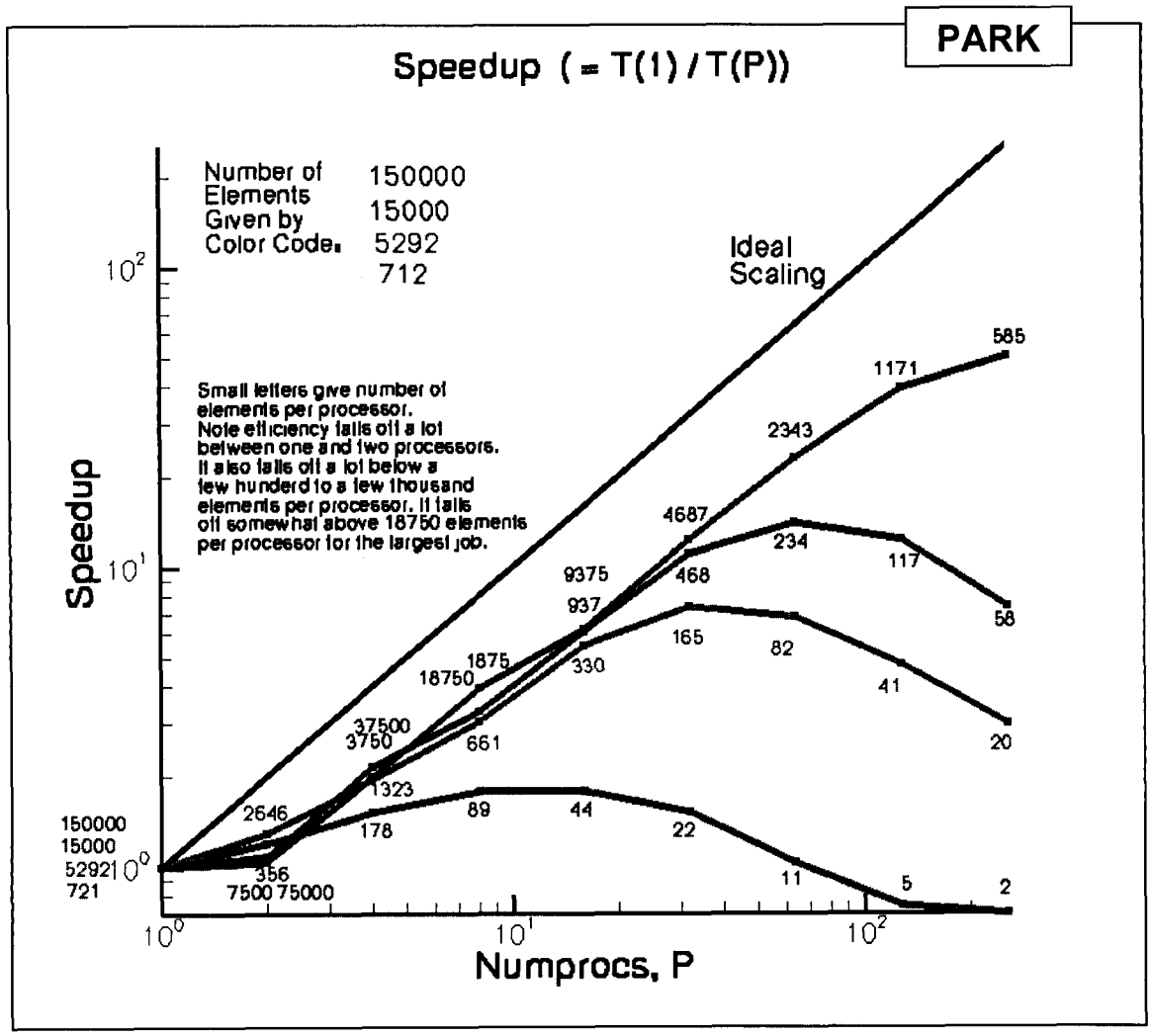

Figure 3b. PARK: Speedup. While two processors do not always run substantially faster than one, the larger problems examined make excellent use of additional processors. Generally one must use at least a few hundred elements per processor to obtain reasonable speedup.

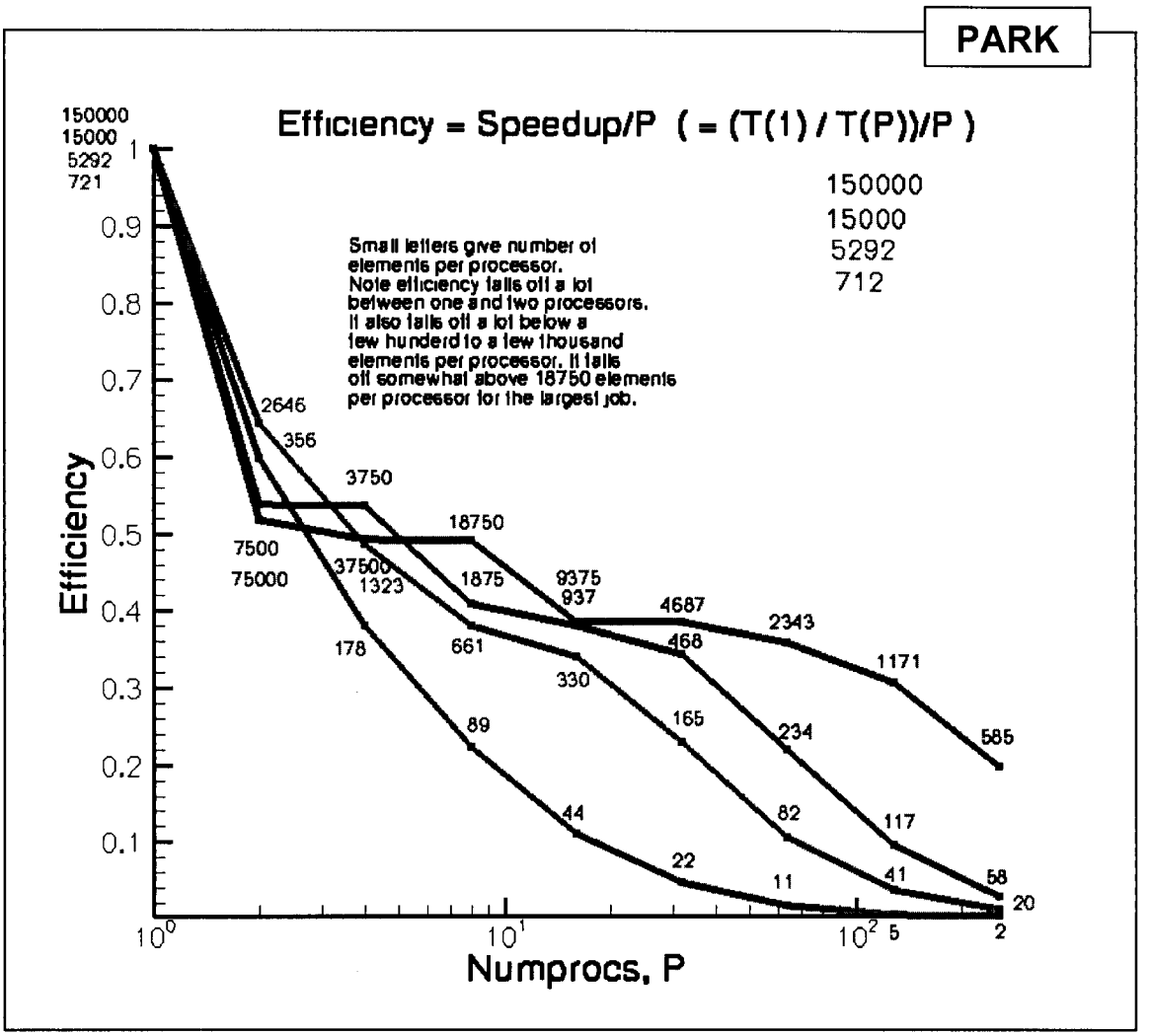

Figure 3c: PARK: Efficiency. Ideally 16 processors would result in a speedup of 16 (for perfect algorithms and instantaneous communication time); here we magnify the degree to which we fall short of that. The drop from 1 to 2 processors may be due to resource contention and will be studied further. The remaining portion of the curves shows reasonably high relative efficiency for the largest problems, especially for cases where several hundred elements reside in each processor.

The scaling data show that not much speedup is gained by going from one to two processors. Further work to understand this behavior could allow us nearly a factor of two in efficiency in future runs. For the largest job (150,000) elements, the efficiency and overhead are nearly constant from 2-8 processors. Efficiency falls off between 8 and 16 processors and is constant from 16-32 processors. For 64, 128 and 256 processors, efficiency falls off gradually. This falloff is

presumably due to an insufficient number of elements per processor, the numbers being 2343, 1171, and 585, respectively, as the plots show. This effect is seen even more dramatically for the jobs with a smaller number of elements because, as the number of processors increases, the number of elements per processor gets so small that a large amount of time is spent communicating between processors. The falloff between 8 and 16 processors on the 150,000-element problem suggests that the optimum number of elements per processor may be about 20,000.

Scientific and Computational Significance

Achieving the First Code Improvement Milestone is significant because it opens the way to run significantly sized problems. For the first time it presents to the scientific community fast parallel codes that allow creating simulations of the entire earthquake cycle on a fault in a 3D model that uses the most accurate description of fault friction, rate and state friction, and the quasi-dynamic radiation damping approximation to full elastodynamics. We now have the potential for greatly increasing the number of elements that can be included in the model over what could be done in the past.

Enough elements can now be used that is it possible to represent a reasonably sized fault with elements that are small enough that they can properly represent the behavior of a continuum. Larger numbers of elements also allow occurrence of earthquakes with a large range of sizes in the simulation. It will now be possible to simulate small earthquakes occurring in isolation and ones that cascade or grow into larger ones. It is currently not understood what causes small earthquakes to grow into large ones or stop at small events. Hence, these new simulations should be key to understanding earthquake rupture processes.

This could help gain an understanding of whether patterns of microseismicity might be used to help predict earthquakes. The attainment of this milestone not only represents an advance in our computational ability to simulate earthquakes, but will allow us to understand the earthquake process better by creating simulated data sets that can be compared with data on real earthquakes. The attainment of the next milestone (Second Code Improvement) will involve increasing the efficiency of the code in other ways, now that the parallel implementation has been achieved, and this will allow even larger and more realistic simulations to be run.

Computationally, we note this is the first simulation of an earthquake fault using the Fast Multipole technique, using a library originally developed for astrophysical, gravitationally-interacting bodies. As shown in Figure 4, this technique leads to enormous savings over traditional full-interaction methods. Note that the Multipole method results in an improvement in scaling on a single processor, and that the performance is continually increased by employing additional processors (for sufficiently large problems).

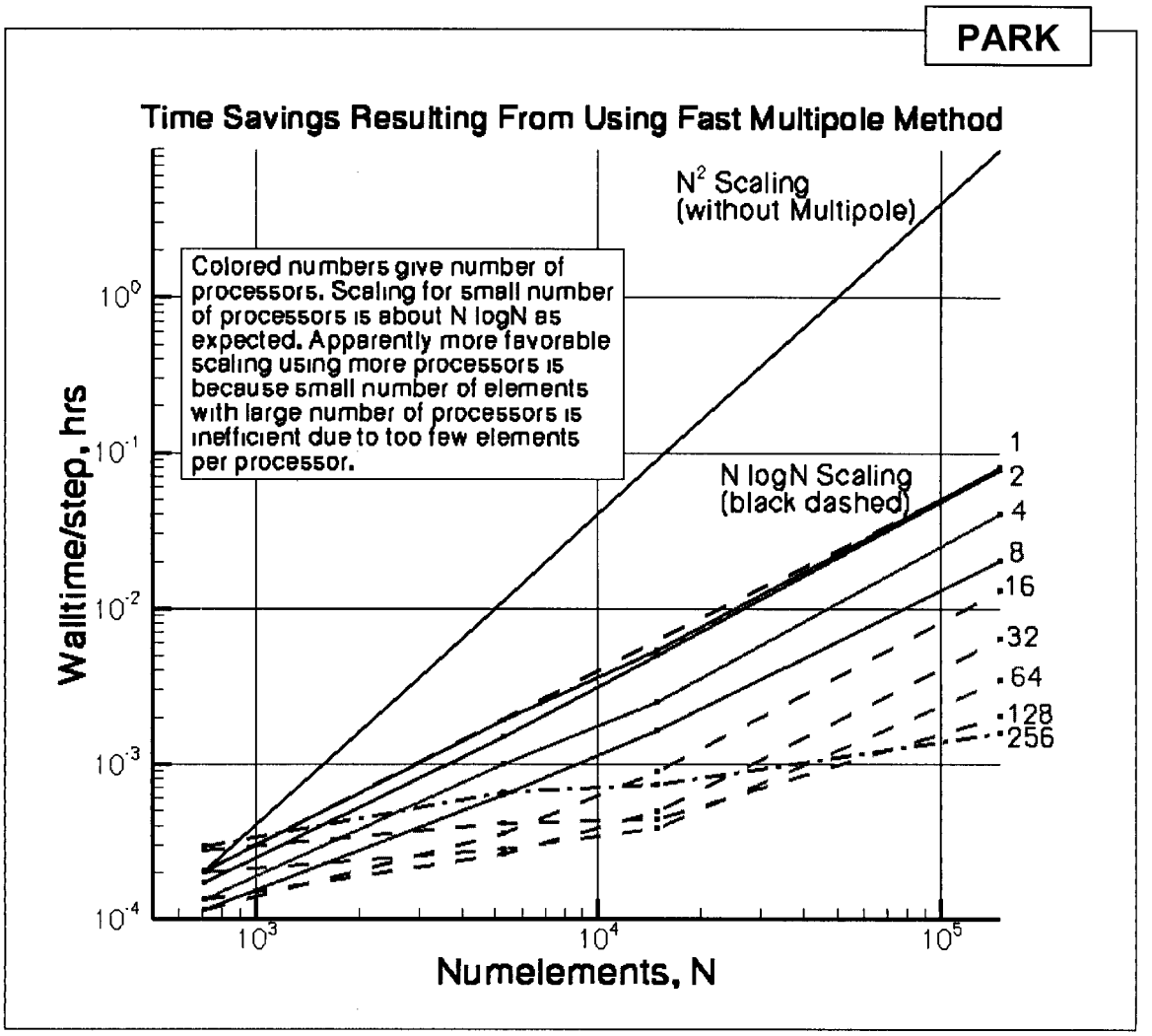

Figure 4: PARK: Scaling of time required per time step using traditional N squared full interaction method, the Multipole method on one processor, and the parallel improvement found with additional processors (to 256).

Simulation details

All of the necessary material that describes the First Code Improvement Milestone can be found within the appropriately named subdirectories under the **1st_Code_Improv_Milestone** directory in the public server at http://www.servogrid.org/slide/GEM/PARK (or on NASA machines *turing* or *chapman*). Instructions are included that will allow duplication of the results.

Included in the **in** and **out** directories are all the materials from the First Code Improvement Milestone run with 150,000 elements and 256 processors for 5,000 time steps. For code testing purposes on one's own system it is useful to set the number of time steps in the prk.dat.150003 file to a smaller number than 5,000 for the initial run; even 2 would be reasonable for the first run.

The materials in these directories include:

"Milestone_Certification_Data.txt" - a file that gives the time required for the First Code Improvement run and describes various parameters of the run.

"README-setting_up_input_files.txt" - a file that tells one how to understand the input files including an explanation of how the elements are created from the input files.

README-Compile.txt - a file that tells how to create both the Multipole library and the PARK fault files using the appropriate Makefiles.

in - a directory that contains the input files that were used in the First Code Improvement run.

out - a directory that contains the output files that were generated in the First Code Improvement run.

src-bin - a directory that contains the PARK and related fault application files used in the First Code Improvement run. The versions of this directory on *turing* and *chapman* also have the object files and executable binary file (named "park").

Chapman is described at http://www.nas.nasa.gov/About/Profile/resources.html. Particulars relevent to this run (current as of November 2003) are:

> *Chapman*, an SGI Origin 3000, is currently the only 1,024-processor single-image, shared-memory system in existence, with one operating system and a single address space. Chapman will become a major component of the IPG (Information Power Grid), and is currently being used to demonstrate that applications can scale to 1,024 processors on this machine. The system has 128 gigabytes of main memory, and 2 terabytes of FC Raid disk storage.

PARK References

Salmon, John K, and Michael S. Warren, Parallel out-of-core methods for N-body simulation. In Michael Heath, Virginia, Torczon. et. al., editiors, *Eighth SIAM Conference on Parallel Processing for Scientific Computing,* SIAM, 1997.

Michael, S. Warren, John K. Salmon, Donald J. Becker, M. Patrick Goda, Thomas Sterling, and Gregoire S. Winckelmas. PentiumPro Inside: I. a treecode at 430 Gflops on ASCI red, II. Price/performance of $50/Mflop on Loki and Hyglac. In *Supercomputing, '97*, Los Alamos, 1997, IEEE Comp. Soc.

Code Description – GeoFEST

GeoFEST simulates stress evolution, fault slip and plastic/elastic processes in realistic materials. The products of such simulations are synthetic observable time-dependent surface deformation on scales from days to decades. Scientific applications of the code include the modeling of static and transient co- and post-seismic Earth deformation, Earth response to glacial, atmospheric and hydrological loading, and other scenarios involving the bulk deformation of geologic media.

Diverse types of synthetic observations will enable a wide range of data assimilation and inversion techniques for ferreting out subsurface structure and stress history. In the short term, such a tool allows rigorous comparisons of competing models for interseismic stress evolution, and the sequential GeoFEST system is being used for this at JPL and UC Davis. Parallel implementation is required to go from local, single-event models to regional models that cover many earthquake events and cycles.

GeoFEST uses stress-displacement finite elements to model stress and flow in a realistic model of the Earth's crust and upper mantle in a complex region such as the Los Angeles Basin. The model includes stress and strain due to the elastic response to an earthquake event in the region of the slipping fault, the time-dependent viscoelastic relaxation, and the net effects from a series of earthquakes. The physical domain may be two or three dimensional and may contain heterogeneous materials and an arbitrary network of faults. The physics models supported by the code include isotropic linear elasticity and both Newtonian and power-law viscoelasticity via implicit/explicit quasi-static time stepping. In addition to triangular, quadrilateral, tetrahedral and hexahedral continuum elements, the program supports split-node faulting, body forces and surface tractions.

Algorithm

GeoFEST reads in a tetrahedral mesh and information on boundary conditions, faulting events and variations in time. The equilibrium conditions are computed based on solution of the elastostatic equations through the finite element method. Then viscoelastic evolution of the stress field is computed based on an implicit technique applied on a series of time steps.

Numerical Methods

Details are found in the GeoFEST User's Guide, which is posted at:

http://www.openchannelsoftware.org/projects/GeoFEST and

http://www-aig.jpl.nasa.gov/public/dus/quakesim/GeoFEST_User_Guide.pdf.

In brief, the elastostatic solution is computed using standard elastostatic finite elements, as found in (e.g.) Hughes. The sparse positive definite system of equations is solved by the Diagonally Preconditioned Conjugate Gradient (DPCG) method.

For the viscoelastic time steps we form a modified stiffness matrix and right-hand side terms according to the method of Hughes and Taylor (1978), which again results in a positive definite system. Each time-step solution is found by the same DPCG function.

Slip on faults is accommodated by a split node technique (Melosh and Raefsky, 1981) that modifies the right-hand side of the matrix system at nodes local to the fault that slips.

Parallel Implementation

We have linked GeoFEST to the Pyramid library (http://www.openchannelsoftware.org/projects/Pyramid) and rely on Pyramid functions for parallel domain partitioning and communication between nodes. This results in one change in processing, and some changes in the code.

There is now a preprocessing step before GeoFEST is run on parallel machines. This is a command-line invocation of an application "gfmeshparse", which derives a full connectivity description from the GeoFEST input file for PYRAMID's use. Note that in the GeoFEST-4.3P download, this application is called "meshgen."

Changes to GeoFEST are chiefly in the use of PYRAMID. PYRAMID is used for partition information, resulting in each processor having a compact segment of the finite element domain for its formation of the local part of the stiffness matrix and solution. PYRAMID is also used for combining these local solution estimates and conjugate gradient vectors at each iteration within of each time step using the "globalize" function that combines the local information and ensures each processor has valid data. A final merging within PYRAMID allows GeoFEST to write a single result file with every node and element represented correctly and uniquely.

Figure 5 shows how time is spent on four processors (stacked vertically) during the Conjugate Gradient iterations. Two iterations are shown, with blank (black) areas indicating the domination of numerical calculation. Red shows the "WAITALL" state that results when some processors finish local operations earlier than others and the matrix-vector product is combined across processors that have adjacent partition information (arrows show communication among these partition-adjacent processors). Violet indicates the "ALLREDUCE" function of parallel communication that is required to globally combine parts of a vector dot product.

Beyond the interest in seeing the fingerprint of a parallel conjugate gradient operation, this image shows the dominance of computation over communication time (black >> colors), and the acceptable (but not perfect) load balance of the partitioned work.

GeoFEST

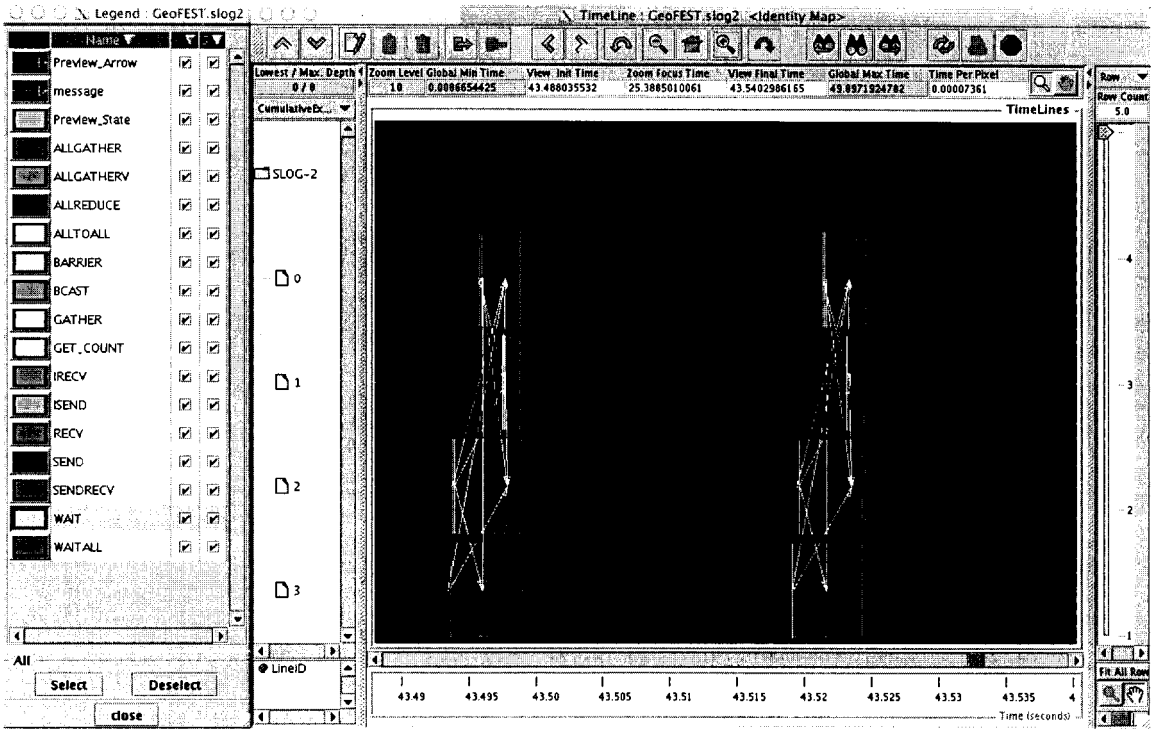

Figure 5: GeoFEST: Fine-detail image of the portion of a GeoFEST run that represents most of the computer time, indicating good parallel performance. Four processors are represented along the vertical axis, and time is represented along the horizontal axis (a 50 millisecond time clip is shown above). Two iterations of the Conjugate Gradient algorithm, of the thousands of iterations making up this simulation, are shown. Three features indicate this algorithm will scale to very large problems: 1) The computational load (thin horizontal turquoise line on black background) is about the same for each processor. 2) The time spent in synchronization and communication (red and violet) is a small fraction of the total. The white arrows indicate the inter-processor communication paths among processors where such communication occurs only as needed. (This explains why some processors spend more time in synchronization than others even though the fraction of time is small.) 3) (not visible in this plot) the fraction of time spent in communication does not grow when problem size grows proportional to the number of processors used.

Documentation

The GeoFEST users guide can be found at
http://www-aig.jpl.nasa.gov/public/dus/quakesim/GeoFEST_User_Guide.pdf

The GeoFEST code and validation case may be downloaded from:
http://www.openchannelsoftware.org/projects/GeoFEST
(follow the "GET IT!" link).

Two files should be downloaded:

- the "GeoFEST User's Guide" (a pdf file, GeoFEST_par_5P.pdf) and
- the "GeoFest 4.3p" (a compressed tar file, GeoFEST-4.3p.tgz).

Another helpful link from the download area is "GeoFEST example" in the left margin under "Additional resources." This link produces an html interactive guide to a simple GeoFEST input file, which illuminates the input format for GeoFEST.

The User's Guide covers the following. The Introduction describes the use of the finite element method for stress and deformation simulation for models of earthquake faults that include many of the complexities of crust and mantle materials. The Features section describes the kinds of physics and boundaries currently supported in GeoFEST. The Theory of Operation section covers the mathematical and computational basis of the GeoFEST simulations, including the mathematics of viscoelastic mechanics, the finite element formulation, the implicit time-stepping scheme, the split-node implementation for faults, and the basis of parallel computation. The Input/Output section describes the formats and meanings of the parts of the relevant files. The section titled Running GeoFEST includes compilation details and parallel execution. There is an annotated sample 2D input file (this corresponds to the "GeoFEST example" interactive link from the GeoFEST Open Channel page, mentioned above). Finally there are two appendices. The first provides flow charts describing the basic organization of the GeoFEST code at the source level, including how GeoFEST links Pyramid calls for parallel operation. The second describes all GeoFEST functional routines, organized by source file.

The compressed file "GeoFEST-4.3p.tgz" may be unpacked using "tar xzf GeoFEST-4.3p.tgz" (on UNIX systems) or the equivalent. This creates a directory **GeoFEST-4.3p** which contains subdirectories **geofest**, **MeshGen**, **Pyramid**, and a text guide "README." Follow the directions in "README" to complete the download and configure the GeoFEST compilation for your machine. One should note that additional libraries are necessary (Pyramid and ParMetis, both of which are freely distributed at the links listed in "README"), and a soft link must be made to configure GeoFEST with Pyramid. (The entire process should take just a few minutes). Note that several example files named "Makefile.*" are given in the **geofest** directory to support various systems and compilers. These may be used as examples for systems not yet supported (usually new compilers or parallel systems are a matter of choosing appropriate compiler flags and incorporating these into the make system).

The MeshGen directory contains a separate sequential program. In particular one will find FORTRAN 90 source GEN_GeoFEST.f90 and a Makefile with support for several FORTRAN 90 compilers. Successful compilation will result in an executable named "meshgen". It generates additional connectivity information from a provided mesh file, and this information is necessary for running Pyramid-enabled parallel GeoFEST. The MeshGen program must be run

on the GeoFEST input file to create a second file (usually with matching prefix to the GeoFEST file and with suffix ".jpl") containing additional mesh information. Then both files are used in a subsequent parallel GeoFEST simulation. This use is described in the subsection "Running the GeoFEST parallel version" in the User's Guide, repeated here for convenience:

> "Running the parallel version of GeoFEST is very similar to the sequential version, with two main differences. The first is that a .jpl file is required in addition to the regular GeoFEST input file in order for the parallel version to successfully execute. This file includes auxiliary geometry data needed by Pyramid to partition the mesh into subdomains. The second is that the output directory path must be specified if the user does not want the output to be written in the ./ directory (which is the default).
>
> "One additional step must be taken with the input for the parallel version. The user will use the meshgen program to convert the regular GeoFEST input into an input file that the parallel code can use (with the .jpl extension). To invoke this program the user simply enters "meshgen", and is interactively prompted to enter the input filename. (Upon completion, filename gives rise to the new file "filename.jpl.")"

The **Pyramid** directory is empty, and is provided for creating the necessary soft links for the Pyramid library (see "README" in the **geofest** directory).

Under the **geofest** directory is a subdirectory **validation**. The "README" file there describes using the meshgen program followed by GeoFEST, followed by a script "summa.pl" that writes a summary of the generated output file, allowing the user to check the correctness of the locally built system. The simulation uses the local file test.dat, which causes GeoFEST to perform ten time steps on the Landers three-fault geometry meshed with about 350,000 elements. It takes a few minutes to run.

Scaling Analysis

GeoFEST processing for the benchmark case naturally divides into three phases:

- Input, including reading the mesh file and creating the mesh partition data for each processor;
- Elastic, where the initial equilibrium state is computed;
- and Viscoelastic evolution, where a sequence of time steps follow the relaxation of shear stresses in viscous material.

Figure 6 shows where time is spent for the milestone case (64 processors, 1.4 million elements, 1,000 time steps)[2]. The preponderance of time is taken up by the viscoelastic time step portion of the code. It is clear that unless many more processors are used (or many fewer time steps of simulation), the scaling of the viscoelastic steps determines the time one may expect for a simulation to complete.

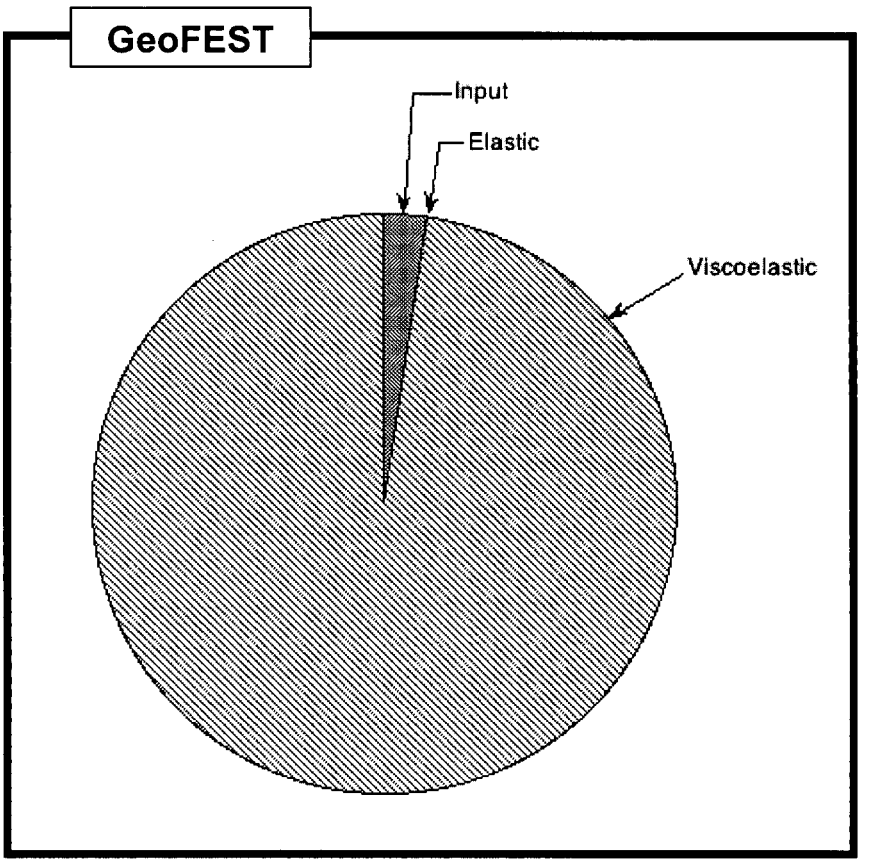

Figure 6: Fraction of total wallclock time for each phase of the simulation (Input, Elastic initial solution, and the Viscoelastic 1000 time steps) for the 1.4 million finite element Landers mesh on 64 processors of *Thunderhead.*

A single Viscoelastic time step consists of construction of a stiffness matrix (and right-hand side), followed by solution of this matrix by the parallel conjugate gradient technique (many iterative steps). So the Viscoelastic segment of Figure 6 represents 1,000 time steps, each consisting of roughly 200 iterations of the sparse solver. One iterative step consists chiefly of a sparse distributed matrix vector multiply and three distributed vector dot products (plus some scalar-vector operations that may be neglected). In Figure 5, the portions represented by

[2] For 8 of the 1,000 time steps, a snap-shot of full displacement data was captured to a file (as wad done in the baseline case for Milestone E).

turquoise lines on black are the local operations for the sparse matrix-vector product (the finite element domain has been partitioned among the processors in contiguous pieces). The red bars indicate synchronization and combining of the solutions where nodes are shared among adjacent processors. The violet represents the synchronization and communication that combines local dot products into full values across all processors.

The operations required for construction of the stiffness matrix are small. If we represent the number of elements as N_{elts}, and number of iterations as N_{iters}, stiffness construction operations number:

$\sim 3500 * N_{elts} = \sim 5$ billion for the $N_{elts} = 1.4$ million

compared to the operations required for a

$N_{iters} = \sim 200$ solution,

which has operation count

$\sim 300 * N_{iters} * 300 * N_{elts} = \sim 80$ billion.

Data for Scaling Runs for GeoFEST				
Number Processors	**Number Elements**	**Viscoelastic Ops**	**Wallclock (s)**	**Operations/ wallclock (s)**
4	82384	2.03E+12	2821.2	7.20E+08
16	82384	2.03E+12	738.9	2.75E+09
64	82384	2.03E+12	252.3	8.05E+09
16	3.53E+05	1.38E+13	4912.7	2.82E+09
64	3.53E+05	1.38E+13	1229.7	1.12E+10
64	1.40E+06	8.11E+13	7247.6	1.12E+10

Table 3: Table 3 shows the problem sizes and run times of GeoFEST (for the viscoelastic simulation phase) on *Thunderhead* using different numbers of processors. In order to make this scaling study, the milestone problem was reduced to 5 time steps (rather than 1,000)

We show in Table 3 the time for completing the time step portion for several short (5 time step) runs of varying size, on *Thunderhead*. Problem sizes were constructed to approximate a constant number of elements in each processor for 4, 16, and 64 processors. Figure 7 shows the speed of solution for these cases: "work" is taken to be the operations involved in the iterative steps. This plot proves communication overhead is not growing faster than computational time as we scale the problem size with number of processors. The result is excellent scaling. Problems with > 4,000 elements per processor see negligible parallel

overhead, and even smaller problems show substantial speed up with many processors.

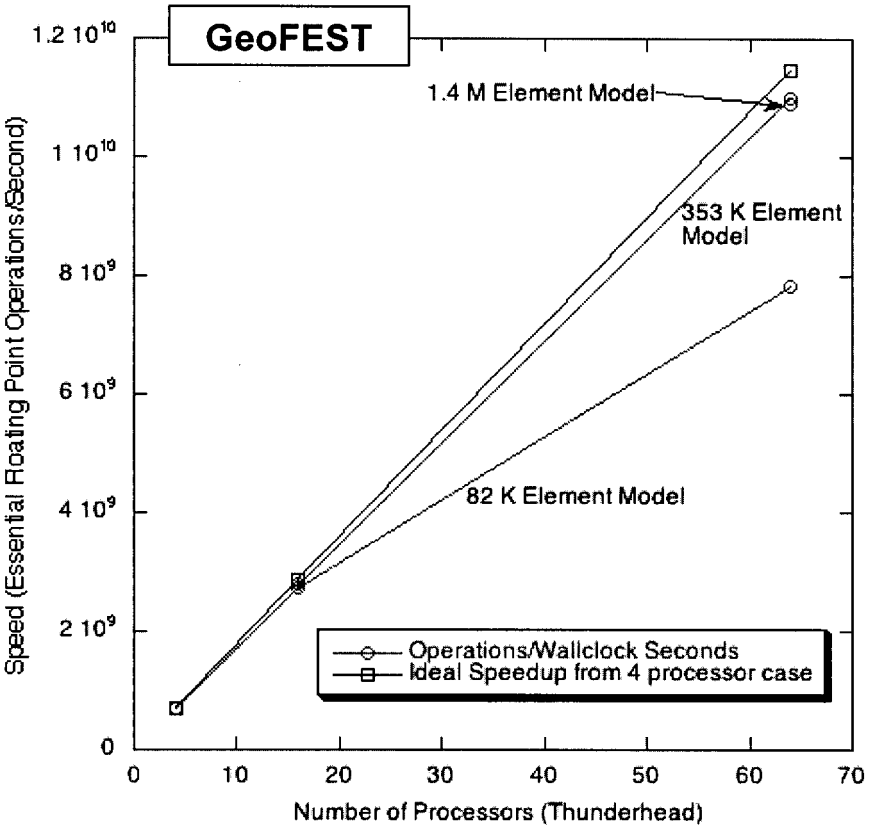

Figure 7: GeoFEST: Scaling of work (on linear scales) in GeoFEST time-step function with number of processors on three sizes of problems (on *Thunderhead* cluster computer, GSFC). Blue indicates ideal scaling (from 4 processors). Expressing work in operations/wallclock time allows comparison of sizes in a single plot.

Figure 7 demonstrates that the GeoFEST scaling (on the dominating time-stepping loop) is excellent. Sufficiently large problems (353,000 and especially 1.4 million elements shown here) have trivial parallel overhead on 64 processors.

More complete scaling analysis considers the time for problem set-up, elastic solution, and writing results to files. Benchmark-type problems with 1,000 time steps are dominated by the scaling behavior shown above, for the example problem (1.4 Million elements, 64 processors).

Scientific and Computational Significance

The baseline problem was modeled after the Northridge earthquake (single thrust fault). Analysis of individual earthquake events with attention to their geographic settings is a long-term, important area of simulation and research. But the ability to solve domains with millions of elements implies we can simulate regions with multiple faults, such as the Los Angeles basin. Interactions among slipping faults and possible emergent structures from these nonlinear interactions appears to be the next advance in forecasting earthquake risk. This is a new and promising area for simulation, data comparison and testing of concepts. Many kinds of simulation codes are beginning to be employed for this new kind of work across the earthquake community, but a finite element code has close to the greatest degree of flexibility in including the effects of realistic structures in the Earth in a heterogeneous domain. We expect this improved GeoFEST will have unique value in validating these other simulations and determining when other multiple-fault models are leaving out too many material effects.

Computationally, we have demonstrated efficient parallel Conjugate Gradient solutions for 3-D faulted-system finite elements, and linked our method with the PYRAMID library. The parallel Conjugate Gradient is no surprise, as it has been demonstrated in other domains of physics. But a freely-available source code for faulted domains will be helpful to the US simulation effort. Linking with PYRAMID is a convenient way to handle issues of partitioning and communication. More important, it is a first step to using the PYRAMID functions for parallel adaptive mesh refinement. Adaptive mesh refinement is essential to attaining high-quality solutions to problems with widely varying stress intensities in three dimensions. Parallel mesh refinement has the additional advantage of solving problems with size commensurate with the memory space of massively parallel computers without handling the associated mesh files with solid meshing programs, which are nearly all written for sequential machines. A domain can be described and meshed at a relatively low mesh density, imported to the parallel system, and then key locations in the mesh can be refined to the degree needed, expanding the number of elements by large factors (possibly hundreds or more).

Simulation details

The GeoFEST code and milestone input file are available on the Goddard Space Flight Center machine *Thunderhead*. Building the code on *Thunderhead* is accomplished by following these steps:

Note: Use the INTEL compiler by typing "mpi_env -p intel"

- Get the code from ~nortonc/Milestone/GeoFEST.tgz
- "tar xvzf GeoFEST.tgz" in your home directory ~/
- "tcsh" to use tcsh
- cd ~/GeoFEST/Pyramid-3D/Pyramid-3D/ParMetis/ and type "make"
- cd ~/GeoFEST/Pyramid-3D/ and type "make -f Intel"
- cd ~/GeoFEST/geofest/ and type "make -f Intel"
- type "exit" to leave tcsh

This results in an executable program called GeoFEST in ~/GeoFEST/geofest.

Follow these steps to run the code on *Thunderhead*.

- Start LACE Task manager with "ltmsuper"
- Move executable to your run directory

  cp ~/GeoFEST/geofest/GeoFEST ~/rhome/GeoFEST/geofest/GeoFEST
- Setup input (input.dat and input.dat.jpl for 4, 16 and 64 processors)

  cd ~/rhome/GeoFEST/geofest/

  ln -s /data1/nortonc/Quakesim/GeoFEST/geofest/Visuals/LandersGr4.dat input.dat

  ln -s /data1/nortonc/Quakesim/GeoFEST/geofest/Visuals/LandersGap4.dat.jpl input.dat.jpl

  The 16 processors case files are LandersGr16.dat and LandersGap16.dat.jpl

  The 64 processors case files are LandersGr64.dat and LandersGap64.dat.jpl
- Request processors and run the code

  cd ~/GeoFEST/geofest/

  ltmbegin -n 4 -m 120

  ltmpi -p 1 GeoFEST input.dat /data1/<your loginid>/<output directory>

  OR

  ltmpi -p 1 GeoFEST input.dat /data1/<your loginid>/<output directory> >& /data1/<your loginid>/OUT.log

  Note: -n means number of processors

-m means number of minutes requested

-p 1 means use one CPU per node (rather than 2 CPUs per node which is the default)

For the16 processor case use -n 16 -m 300

For the 64 processors case use -n 64 -m 360

Follow these steps to interpret the results.

- When the code runs, some welcome and diagnostic messages will be printed to the screen.
- Visualization and the conjugate gradient history files will be written in /data1/<your loginid>/<output directory>.
- For scalability timings, capture ("grep -i pyramid OUT.log") the output that looks like this (the bracketed items are explanatory and the data will be on the screen and also in OUT.log if you chose to save it as suggested above):

  PYRAMID Current Time: xxxx.xx [A. Loading Data Start]

  PYRAMID Current Time: xxxx.xx [B. Elastic Soln Start]

  PYRAMID Current Time: xxxx.xx [C. Time Stepping Start]

  PYRAMID Current Time: xxxx.xx [D. Program Termination]

  The total runtime will be in step D.

- In order to examine scalability, the key number is the average time per iteration in the time stepping stage. Since the problem size grows somewhat proportionally to the number of processors in a scaled fashion 4, 16, 64, this number should look roughly constant across these problem sizes.
- Compute the average time per iteration as (D-C)/X where X is the total number of iterations for each problem size

  4 PE case: X = 83,256

  16 PE case: X = 132,429

  64 PE case: X = 195,914

As an aside, the baseline case ran in about 13 hours for a problem about the same size as the 4 PE case above, so for scalability relative to the baseline case the 4 PE case would have to run in less than 3 hours. Our experience has been that the 4 PE case can run in 1 hour (dual cpu mode) or less (single cpu mode).

Thunderhead is described at:

http://newton.gsfc.nasa.gov/thunderhead/index.htm. Particulars relevant to this run (current as of November 2003) are:

> *Thunderhead* is a 512-processor Commodity Cluster located at NASA/Goddard Space Flight Center. It features 512 2.4Ghz Pentium 4 Xeons, 256Gb DDR memory, 20Tb disk space and 2.2Gpbs myrinet fiber interconnect.

References

Thomas J. R. Hughes and Robert Taylor, "Unconditionally Stable Algorithms For Quasi-Static Elasto-Plastic Finite Element Analysis." Computers & Structures, Vol. 8, pp. 169-173, 1978.

Thomas J. R. Hughes, "The Finite Element Method: Linear Static and Dynamic Finite Element Analysis." Dover, Publication, INC., Mineola, New York, 2000.

H. J. Melosh and Raefsky, "A Simple and Efficient Method for Introducing Faults into Finite Element Computations." Bulletin of the Seismological Society of America, Vol. 71, No. 5, October 1981.